Een Softwarekwekerij?

Paul Klint

9 juni 2001

1 Inleiding

Nederlands bekendste fokstier—de Skalsumer stier Sunny Boy—is na het produceren van miljoenen "rietjes" sperma, en na het verwekken van een dienovereenkomstig aantal nakomelingen, eind 1997 overleden. De kwekerijen in het Westland zijn bekend om hun bloemen-, fruit- en groententeelt. De kippen- en varkensbedrijven zijn niet meer te tellen. Kortom, het rijtje beroepen dat vaak opduikt om Nederlandse activiteiten te karakterisen is incompleet: de lijst bestaande uit *dominee* en *handelaar* moet zeker uitgebreid worden met *kweker* of *fokker*.

Nu zijn de neveneffecten van al dat gefok en gekweek ook niet mis: de mestoverschotten, de kwetsbaarheid voor ziektes (varkenspest, mond- en klauwzeer), en de afnemende biodiversiteit (ongeveer 1/6 van de Nederlandse melkproductie wordt voortgebracht door nakomelingen van de hierboven genoemde Sunny Boy) zijn maar enkele voorbeelden.

Zijn deze ongewenste neveneffecten niet te vermijden door onze kweekdrift te richten op activiteiten die het milieu niet belasten, zoals, bijvoorbeeld, het kweken van software?

Evolutionaire principes zijn vanuit twee invalshoeken op software toe te passen: als beschrijvingsmechanisme voor de levenscyclus van bestaande software en als methode om nieuwe software te maken. Beide invalshoeken komen nu eerst kort aan de orde. Daarna volgen enkele speculaties over de (on)mogelijkheden van softwarekwekerijen.

1.1 Evolutie en bestaande software

Belady en Lehman [1] hebben al ruime tijd geleden onderkend dat softwareontwikkeling een evolutionair proces is. Zij hebben hun waarnemingen vastgelegd in drie wetten:

- *De wet van voortdurende verandering*: een systeem dat gebruikt wordt ondergaat voortdurende verandering totdat besloten wordt dat het goedkoper is om het systeem te bevriezen en te vervangen.

- *De wet van de toenemende entropie*: de entropie van een systeem (de mate van ongestruktureerdheid) neemt toe naarmate de tijd verstrijkt, tenzij expliciete maatregelen getroffen worden om de entropie te reduceren.

- *De wet van de statistisch vlakke groei*: waarnemingen van systeemgroei mogen lokaal gezien willekeurig lijken in tijd en ruimte, maar statistisch gezien betreft het cyclische, zelf-regulerende, processen met goedgedefinieerde lange termijn trends.

Over de evolutionaire aspekten van software-ontwikkeling is nog veel meer te zeggen [2]:

- Verschillende delen van een systeem bevinden zich meestal in verschillende fasen van ontwikkeling, daarom is de naadloze integratie van methode's voor nieuwbouw, onderhoud, en renovatie van groot belang.

- De ontwikkeling van software is te ingewikkeld geworden om nog op een hierarchische, top down, wijze te kunnen controleren. Meer evolutionaire principes lijken daarvoor meer geschikt te zijn.

- Hergebruik is van belang om goede, goedkope, software te kunnen produceren.

- Het samenstellen van systemen uitgaande van bestaande componenten is daarbij een sleuteltechnologie.

1.2 Evolutie en nieuwe software

Uit het werk van o.a. John Koza [3] is bekend dat evolutionaire, Darwinistische, principes ook gebruikt kunnen worden om software te kweken. Uitgaande van een beginpopulatie van willekerige programma's worden steeds de "beste" programma's gekozen voor voortplanting om zo na een aantal generaties te komen tot gekweekte programma's die een gegeven taak zo goed mogelijk uitvoeren. Hierbij spelen enkele begrippen een rol die allereerst toelichting verdienen.

De *zoekruimte* waarin de evolutie plaatsvindt omvat alle individuele programma's die ooit kunnen ontstaan. Het zou in principe mogelijk zijn om hiervoor de pure *tekst* van programma's als uitgangspunt te nemen. Het blijkt echter beter om meer struktuur in programma's te onderkennen en ze als boomstruktuur voor te stellen. Dit vergroot de kans dat bij het muteren en sexueel reproduceren van programma's ook weer zinnige programma's ontstaan; de kans daarop bij puur tekstueel knippen en plakken van programma's is veel kleiner. In ieder probleemgebied wordt van te voren bepaald welke primitieven in een programma voor mogen komen zoals, bijvoorbeeld, rekenkundige bewerkingen, bewegingen van een robotarm, of zetten in spel.

De *initiële structuren* waarmee het evolutieproces start bestaan uit een willekeurig aantal, willekeurig gekozen, programma's.

Een *fitness functie* bepaalt hoe goed een individueel programma erin slaagt om een gegeven probleem op te lossen. Volgens het principe van de "survival of the fittest" zullen uiteindelijk die individuen of hun nakomelingen (programma's) overblijven die het probleem het beste oplossen.

Sexuele reproductie vindt plaats door eerst een paar "fitte" ouders (twee programma's, beide met een hoge fitness score) te selekteren en deze een nakomeling te laten produceren door compositie van delen van beide ouders. Het resulterende kind (weer een programma) wordt toegevoegd aan de volgende generatie en neemt daarin weer deel aan fitness bepaling en reproductie.

Het kweken van een programma is nu bijvoorbeeld toe te passen om uitgaande van een klein aantal waarden van een functie f, de functie zelf te vinden. Gegeven zijn, bijvoorbeeld, de waarden $f(1), ..., f(10)$ en de vraag is om een programma te kweken dat f zo goed mogelijk berekent. De eerste generatie programma's bestaat uit willekeurige programma's opgebouwd uit willekeurige, rekenkundige, operaties. De fitness van een programma wordt bepaald door de mate waarin het voor de inputwaarden $1, ..., 10$ de waarden $f(1), ..., f(10)$ goed berekent. Na enkele tientallen tot honderden generaties zal blijken dat er programma's ontstaan die de waarden van f aardig, maar misschien niet perfekt, benaderen.

Het blijkt dat deze methode van *genetisch programmeren* vooral goed toe te passen is om problemen op te lossen waarvoor geen eenvoudige analytische oplossing bestaat, zoals het modelleren van de beurskoersen of gebruikersgedrag, het oplossen van plannings- en zoekproblemen, en patroonherkenning. De opgeleverde programma's hebben echter een puur functioneel gedrag, d.w.z. ze leveren voor gegeven invoerwaarden een antwoord, maar ze zijn niet in staat om neveneffecten te veroorzaken zoals het opslaan van waarden in een bestand of het fysiek besturen van een apparaat. Dit maakt het voorlopig nog niet mogelijk om "echte" programma's te kweken die, bijvoorbeeld, de administratie van een bedrijf uitvoeren of een fabriek besturen.

2 Hoe zou een softwarekwekerij eruit zien?

Zal een softwarekwekerij bestaan uit hallen met onafzienbare rijen met bakken waarin dampende hompen software liggen te rijpen? Als stallen waarin de software-equivalenten van Sunny Boy staan te wachten tot zij weer hun bijdrage aan de Goede Zaak kunnen leveren? Of ziet de softwarekwekerij eruit als een gewone kas met bloembedden waaruit de softwarelootjes voorzichtig hun kopjes verheffen? Vast niet!

Op dit punt in mijn betoog aangekomen moet ik de lezer die een kant en klare beschrijving van de softwarekwekerij verwacht had teleurstellen. Ik heb alleen een aantal waarnemingen en vooral een groot aantal vragen te bieden die men op weg naar de softwarekwekerij tegen zal komen. We kunnen in elk geval het volgende vaststellen:

- Bij het kweken van dieren en planten gaat men uit van gegeven genetisch materiaal dat verbeterd en vermenigvuldigd wordt.

- Bestaand genetisch materiaal bevat een grote diversiteit die het resultaat is van een zeer lange evolutie. Dit is de reden dat, onder andere, de farmaceutische industrie de tropische regenwouden afstroopt—zolang dat nog kan—op zoek naar planten met medicinale eigenschappen.
- Het genetisch materiaal is opgebouwd uit slechts vier nucleïnezuren die tot lange strengen aaneengeregen zijn.
- Identificeerbare eigenschappen van individuele planten of dieren liggen vast in *genen*, op zich reeksen nucleïnezuren.
- Bij het kweken van programma's, zoals hierboven beschreven, gaat men uit van willekeurige programma's die via natuurlijke selectie verbeterd worden.
- Bij het bouwen van software wordt in toenemende mate gebruik gemaakt van het combineren van bestaande componenten.

Om te komen tot een op evolutionaire principes gebaseerde softwarekwekerij moeten de volgende vragen beantwoord worden:

- Wat zijn de nucleïnezuren van de software? Het spectrum van mogelijkheden omvat (geordend naar toenemende complexiteit):
  - een laagniveau instructie die direct door een computer uitgevoerd kan worden (b.v. het plaatsen van een waarde in een register);
  - een statement uit een programmeertaal (b.v. een toekenningsstatement, een conditioneel statement of een herhalingsconstructie);
  - een procedure uit een programmeertaal (die zelf weer een aantal statements bevat);
  - een module uit een programmeertaal (die zelf weer een aantal procedures bevat);
  - een compleet programma (dat zelf weer een aantal modules bevat);
  - een compleet systeem (dat zelf weer een aantal programma's bevat).
- Hoe kunnen deze software-nucleïnezuren gecombineerd worden? In de natuur worden nucleïnezuren gecombineerd door ze in een lineaire struktuur (streng) aan elkaar te plakken. In software bestaan er, naast het achter elkaar plakken van elementen, nog diverse andere compositiemechanismen: procedure-aanroepen, importeren van modulen, en coordinatietalen voor de dynamische, op workflow principes gebaseerde, koppeling van componenten. Welke bestaande, of nieuw te ontwikkelen, compositiemechanismen zijn het meest geschikt voor software-evolutie?
- Welke "software-genen" komen in bestaande programma's voor en kunnen dienen als uitgangspunt voor een softwarekwekerij? Genen bepalen één specifiek deelaspect van een organisme (kleur van de ogen, lichaamsbouw,

e.d.). Combinaties van genen bepalen complete functionele delen van een organisme (b.v. de ogen). De momenteel gebruikte softwarecomponenten bevatten de complete functionaliteit van een onderdeel van een systeem (b.v. het verwerken van de datum, conversie van valuta, opzoeken van een naam in een tabel). Op deze manier beschouwd zijn de huidige softwarecomponenten groter dan software-genen en zijn ze daardoor vermoedelijk te groot om als basis te dienen voor een softwarekwekerij.

Een belangrijk aspect van de huidige softwarecomponenten is *abstractie*: aan de buitenkant van een component is alleen zijn extern gedrag waar te nemen terwijl zijn interne struktuur verborgen blijft. Bij evolutionaire processen kunnen echter ook mutaties optreden in genen, waardoor er een wijziging ontstaat in hun interne struktuur.

Tenslotte speelt *taal* een cruciale rol bij het vastleggen van software-genen, maar *welke* taal? Het lijkt duidelijk dat hier een programmeertaal-neutrale benadering gewenst is die in elk geval abstraheert van de verschillende programmeertalen waarin gekweekte software uiteindelijk vastgelegd zal moeten worden voor gebruik.

3 Perspectief

We kunnen de evolutionaire aspecten van bestaande software en de evolutionaire methodes voor het kweken van nieuwe software integreren tot een nieuwe visie op software als levend organisme: nieuwbouw komt overeen met het kweken van nieuwe software terwijl onderhoud en renovatie overeenkomen met chirurgische ingrepen in bestaande software die daarna opgekweekt moet worden tot een weer bevredigend werkend exemplaar. Principes van zelfregulatie en -herstel kunnen daarbij van groot nut zijn.

De realisatie van de hier gegeven speculaties over een softwarekwekerij ligt nog jaren vóór ons. Ik hoop echter dat de besproken analogieën kunnen helpen bij de verdere ontwikkeling van concepten die leiden tot meer inzicht in bouw en beheersing van complexe softwaresystemen.

4 Literatuur

1 L.A. Belady en M.M. Lehman, A model for large program development, *IBM Systems Journal*, Vol. 15, No. 1, 1976, p 225-252.

2 P. Klint en C. Verhoef, Evolutionary Software Engineering with Components, te verschijnen in Proceedings *Systems Implementation 2000: Languages, Methods & Tools*, Berlijn, 23-26 Februari 1998.

3 J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.